Curso - Padrões de Projeto Módulo 2: Padrões de Criação

Vítor E. Silva Souza
vitorsouza@gmail.com

http://www.javablogs.com.br/page/engenho
http://esjug.dev.java.net

Sobre o Instrutor

- Formação:
  - Graduação em Ciência da Computação, com ênfase em Engenharia de Software, pela Universidade Federal do Espírito Santo (UFES);
  - Mestrado em Informática (em andamento) na mesma instituição.
- Java:
  - Desenvolvedor Java desde 1999;
  - Especialista em desenvolvimento Web;
  - Autor do blog Engenho – www.javablogs.com.br/page/engenho.
- Profissional:
  - Consultor em Desenvolvimento de Software Orientado a Objetos – Engenho de Software Consultoria e Desenvolvimento Ltda.

Estrutura do Curso

Módulo 1	Introdução
Módulo 2	Padrões de Criação
Módulo 3	Padrões de Estrutura
Módulo 4	Padrões de Comportamento
Módulo 5	O Padrão Model-View-Controller

Conteúdo deste módulo

- Introdução;
- Abstract Factory;
- Builder;
- Factory Method;
- Prototype;
- Singleton;
- Conclusões.

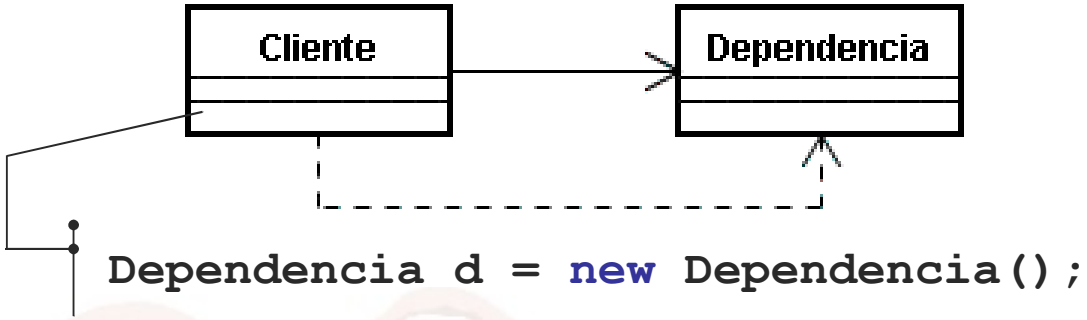
Curso - Padrões de Projeto
Módulo 2: Padrões de Criação

Introdução

Criação de Objetos

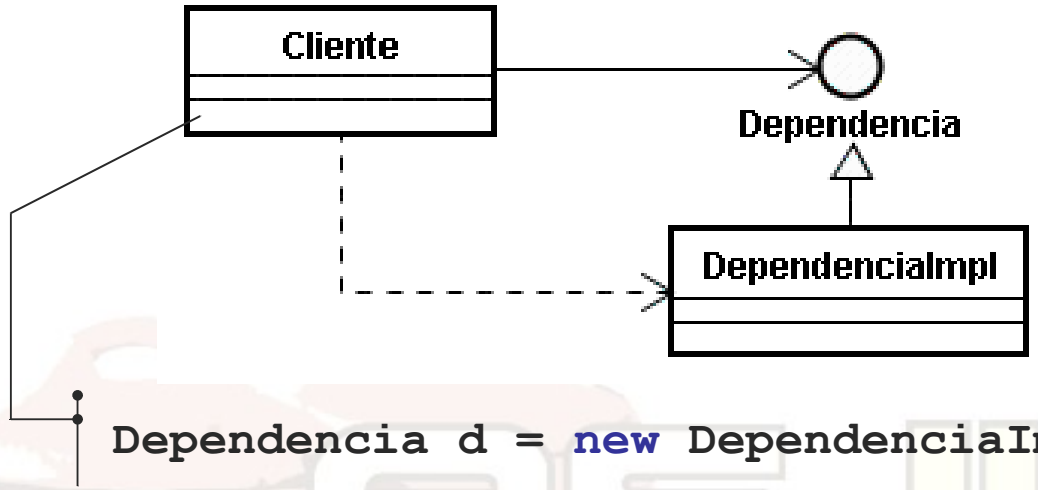
- Em um sistema OO, precisamos criar objetos que irão interagir para construir a solução do problema.

Abordagens para criação – I



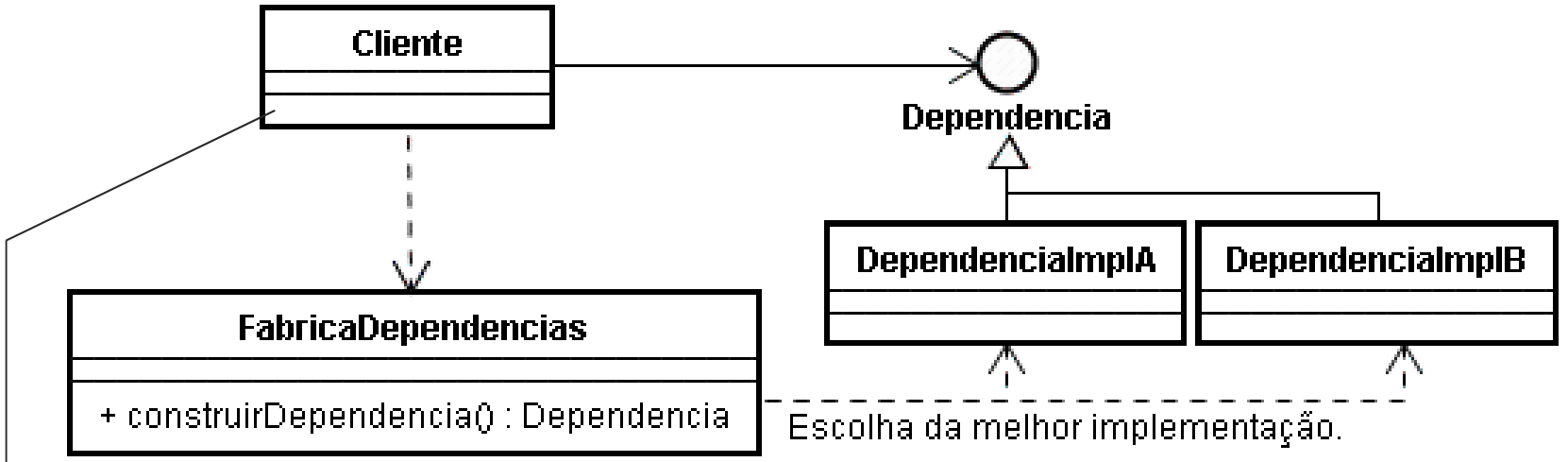
- Solução inflexível:
  - Cliente se refere a uma implementação específica de sua dependência;
  - Cliente constrói diretamente uma instância específica de sua dependência.

Abordagens para criação – II



- Alguma flexibilidade:
  - Cliente já não mais associa-se com uma classe concreta;
  - Porém, instancia a mesma diretamente.

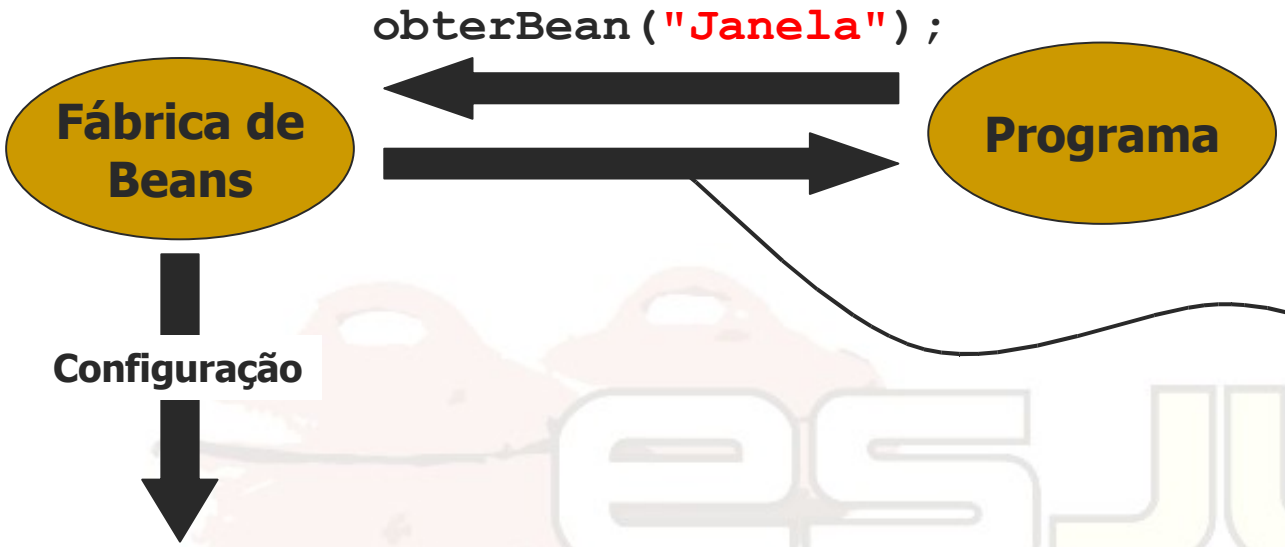
Abordagens para criação – III

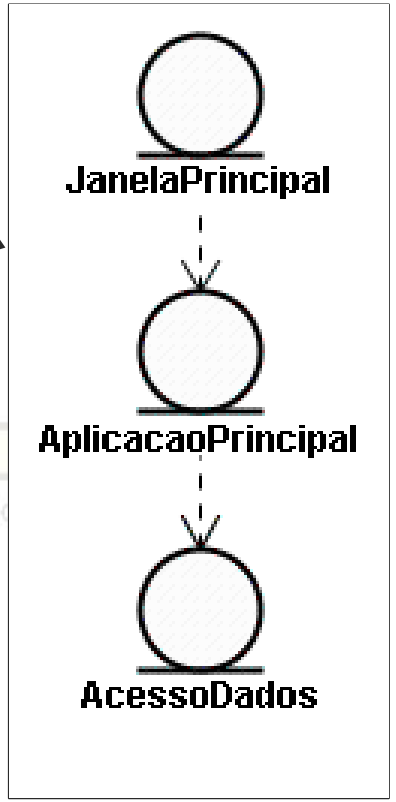



```
FabricaDependencias fabrica = new FabricaDependencias();
Dependencia d = fabrica.construirDependencia();
```

- Maior flexibilidade:
  - A fábrica pode escolher a melhor classe concreta de acordo com um critério: hard-coded, meta-dado, parâmetro, etc.

Injeção de Dependências






```
bean: Janela
  classe: janelas.estiloA.JanelaPrincipal
  dependência: aplicacao = bean Aplicacao
bean: Aplicacao
  classe: aplicacao.nucleo.AplicacaoPrincipal
  dependência: acessoDados = bean AcessoDados
bean: AcessoDados
  classe: persistencia.mysql.AcessoDados
```

Cuidados com a flexibilização

- Maior flexibilização =
  - Menor chance de ter que reprojetar;
  - Menor desempenho;
  - Maior dificuldade de realizar coisas simples;
  - Maior dificuldade de compreender o código.
- Ex.: meta-data hell.

Padrões de Criação

"Abstraem o processo de instanciação, ajudando a tornar o sistema independente da maneira que os objetos são criados, compostos e representados".

Escopo de Classe	Factory Method
Escopo de Objeto	Abstract Factory
	Builder
	Prototype
	Singleton

Curso - Padrões de Projeto
Módulo 2: Padrões de Criação

Abstract Factory

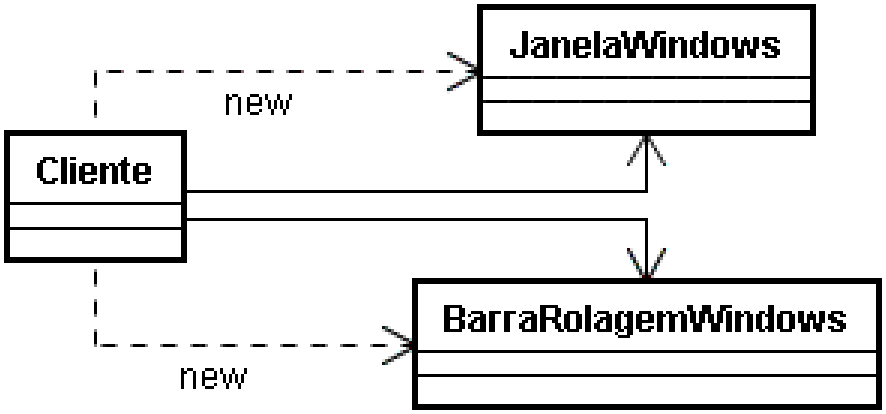
(Fábrica Abstrata)

Criação / Objeto

Descrição

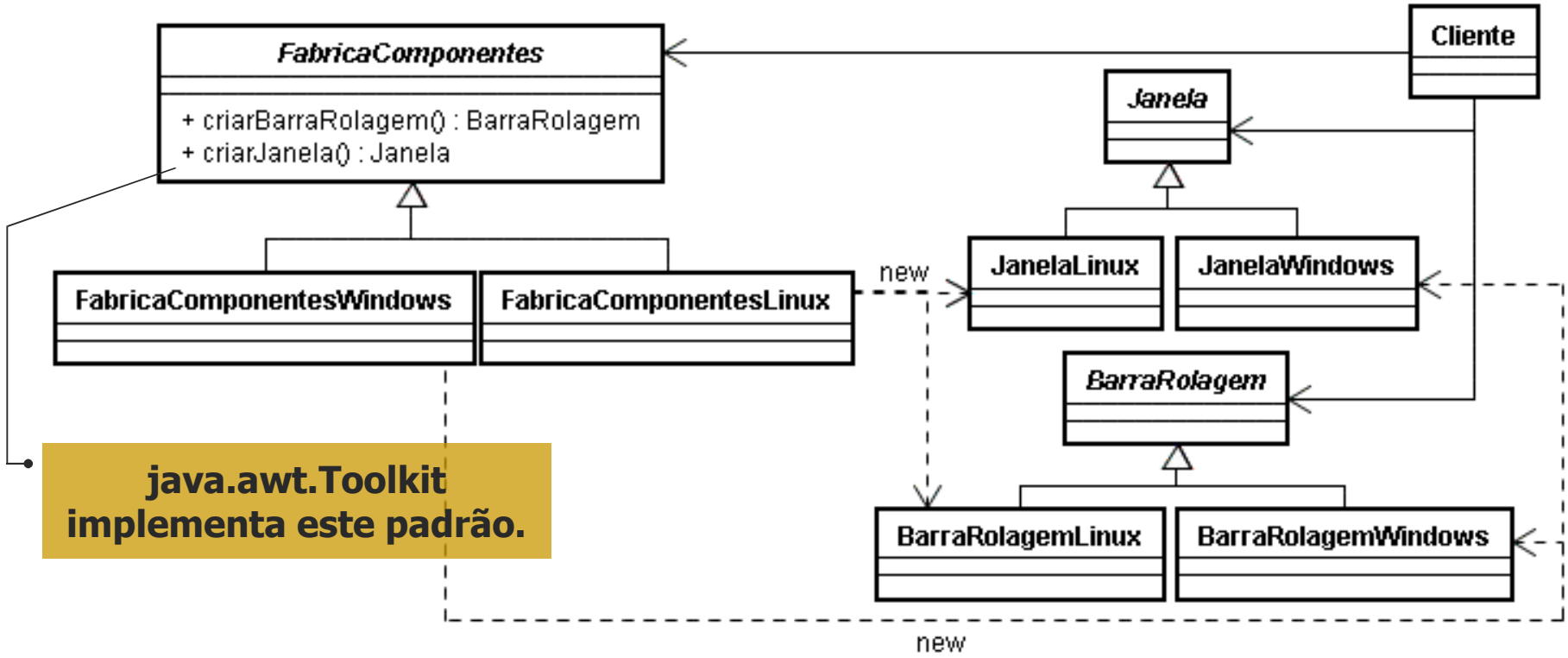
- Intenção:
  - Prover uma interface para criação de famílias de objetos relacionados sem especificar suas classes concretas.
- Também conhecido como:
  - Kit.

O problema



- O sistema acima utiliza APIs específicas da plataforma para criar o "Look & Feel";
- Como desenvolvê-lo já preparado para ser portado para outra plataforma?

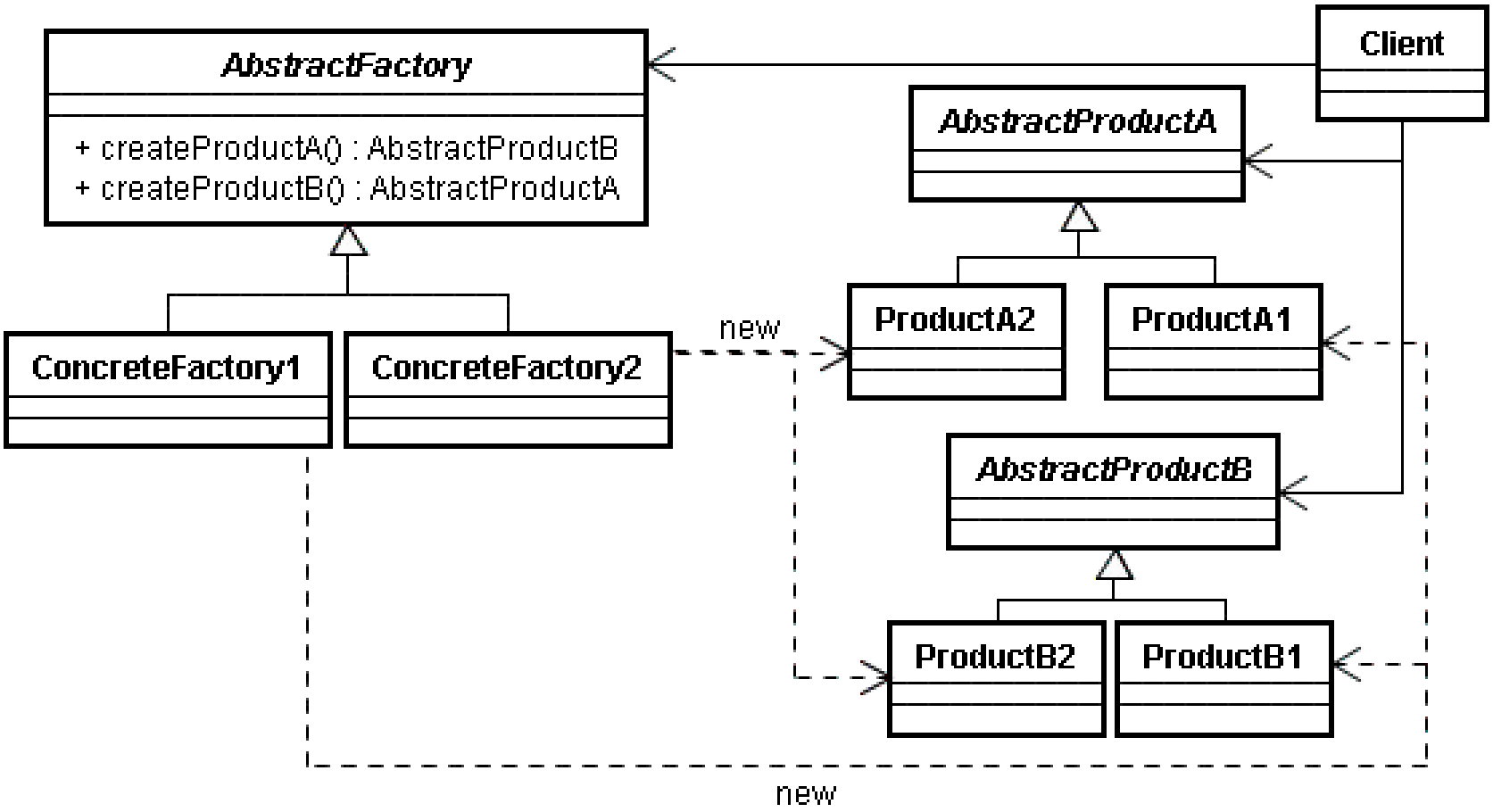
A solução



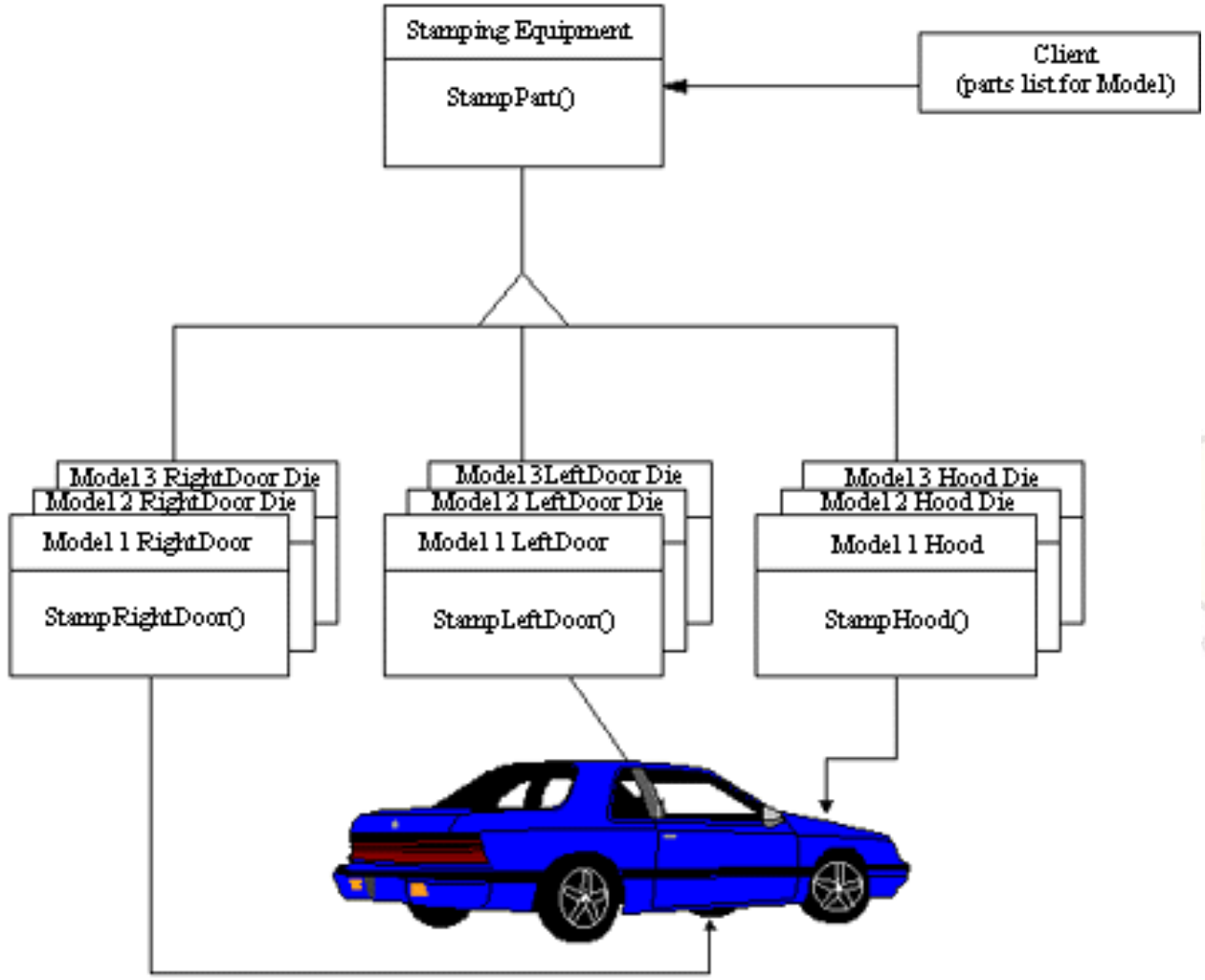
java.awt.Toolkit implementa este padrão.

- Uma das fábricas de componentes se encarregará da criação dos objetos;
- Cliente não conhece as classes concretas.

Estrutura


AbstractFactory
+ createProductA() : AbstractProductB
+ createProductB() : AbstractProductA
Client
AbstractProductA
ProductA2
ProductA1
new
ConcreteFactory1
ConcreteFactory2
AbstractProductB
ProductB2
ProductB1
new

Analogia



Non-Software Examples of Software Design Patterns by Michael Duell
http://www2.ing.puc.cl/~jnavon/IIC2142/patexamples.htm

Usar este padrão quando...

- o sistema tiver que ser independente de como seus produtos são criados, compostos ou representados;
- o sistema tiver que ser configurado com uma ou mais famílias de produtos (classes que devem ser usadas sempre em conjunto);
- você quiser construir uma biblioteca de produtos e quiser revelar apenas suas interfaces e não suas implementações.

Vantagens e desvantagens

- Isola classes concretas:
  - Fábricas cuidam da instanciação, o cliente só conhece interfaces.
- Facilita a troca de famílias de classes:
  - Basta trocar de fábrica concreta.
- Promove consistência interna:
  - Não dá pra usar um produto de uma família com um de outra.
- Criar novos produtos é trabalhoso:
  - É necessário alterar as implementações de todas as fábricas para suportar o novo produto.

Curso - Padrões de Projeto
Módulo 2: Padrões de Criação

Builder

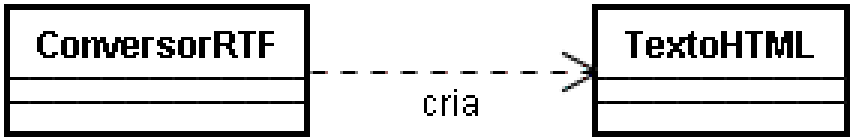
(Construtor)

Criação / Objeto

Descrição

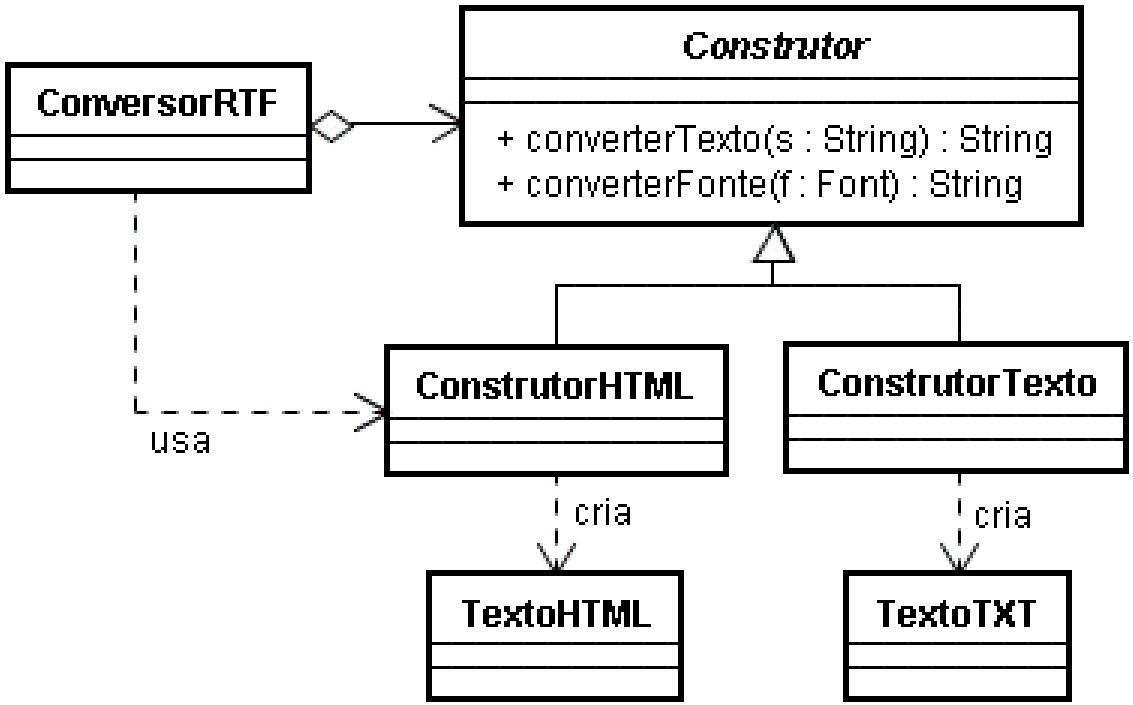
- Intenção:
  - Separar o processo de construção de um objeto complexo de sua representação para que o mesmo processo possa criar diferentes representações.

O problema



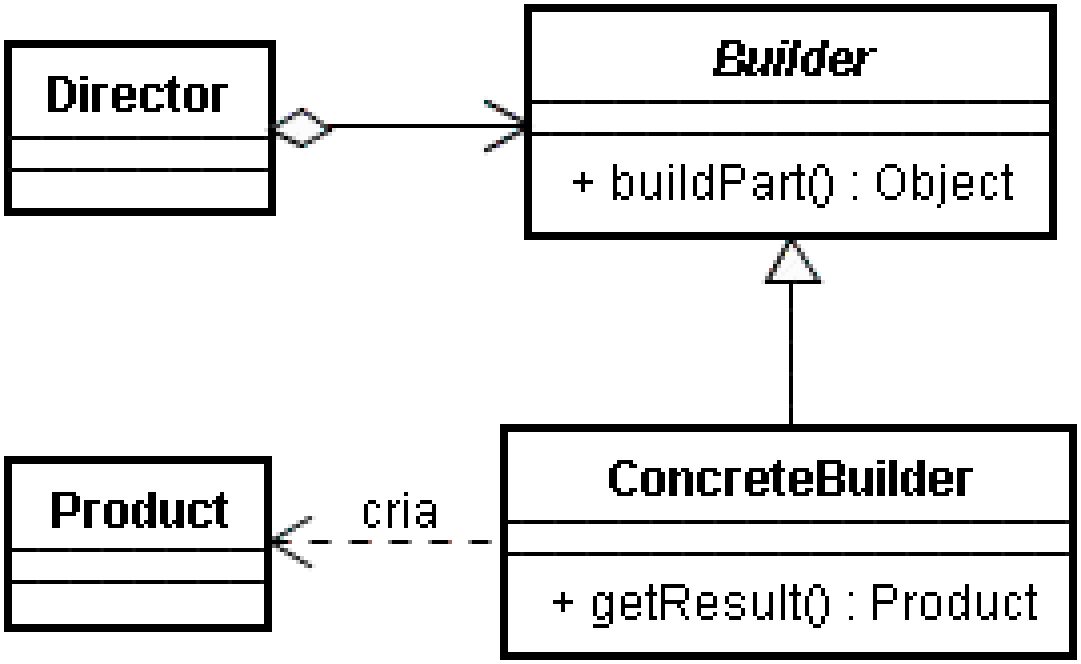
- Um objeto lê um texto em RTF e converte para HTML como produto final de um processamento;
- Como fazer para preparar este sistema para uma eventual mudança de formato (texto puro, por exemplo)?

A solução

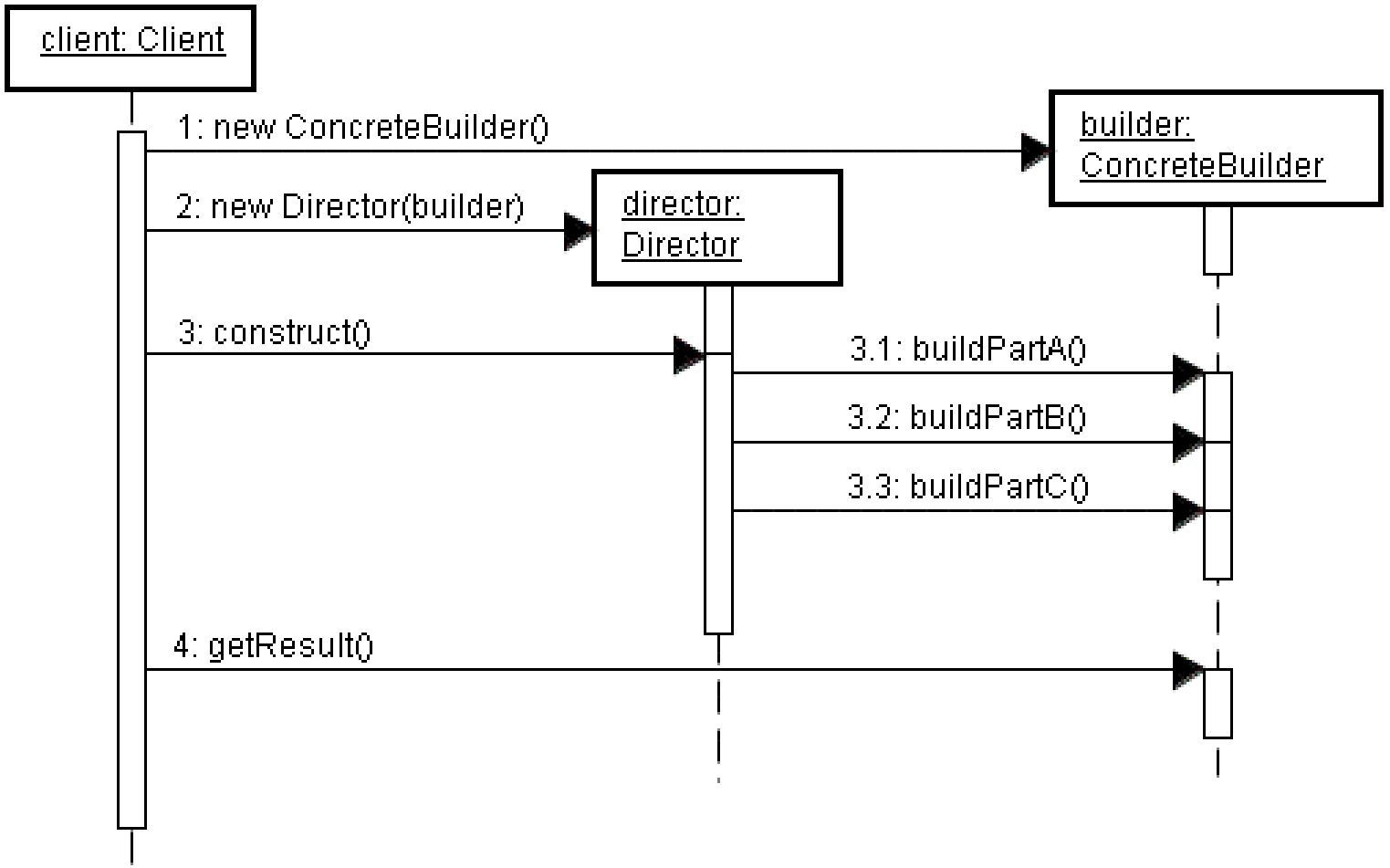


- Para um sistema que mostre o resultado em texto puro, basta trocar o construtor.

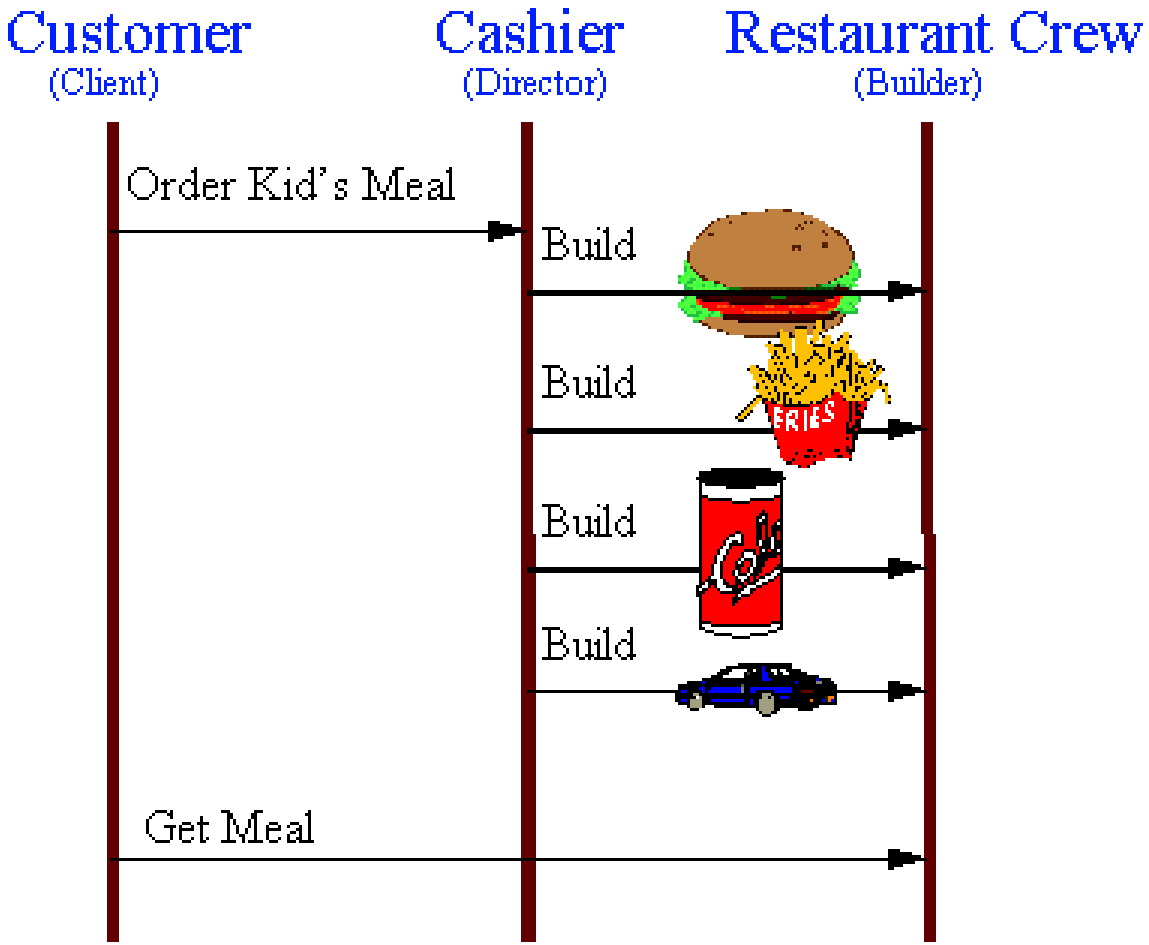
Estrutura


Director
Builder
+ buildPart() : Object
Product
cria
ConcreteBuilder
+ getResult() : Product

Algoritmo


client: Client
1: new ConcreteBuilder()
builder:
ConcreteBuilder
2: new Director(builder)
director:
Director
3: construct()
3.1: buildPartA()
3.2: buildPartB()
3.3: buildPartC()
4: getResult()

Analogia



Non-Software Examples of Software Design Patterns by Michael Duell
http://www2.ing.puc.cl/~jnavon/IIC2142/patexamples.htm

Usar este padrão quando...

- o algoritmo para criação de objetos complexos tiver que ser independente das partes que compõem o objeto e como elas são unidas;
- o processo de construção tiver que permitir diferentes representações do objeto construído.

Vantagens e desvantagens

- Permite que varie a representação interna de um produto:
  - Basta construir um novo builder.
- Separa o código de construção:
  - Melhora a modularidade, pois o cliente não precisa saber da representação interna do produto.
- Maior controle do processo de construção:
  - Constrói o produto passo a passo, permitindo o controle de detalhes do processo de construção.

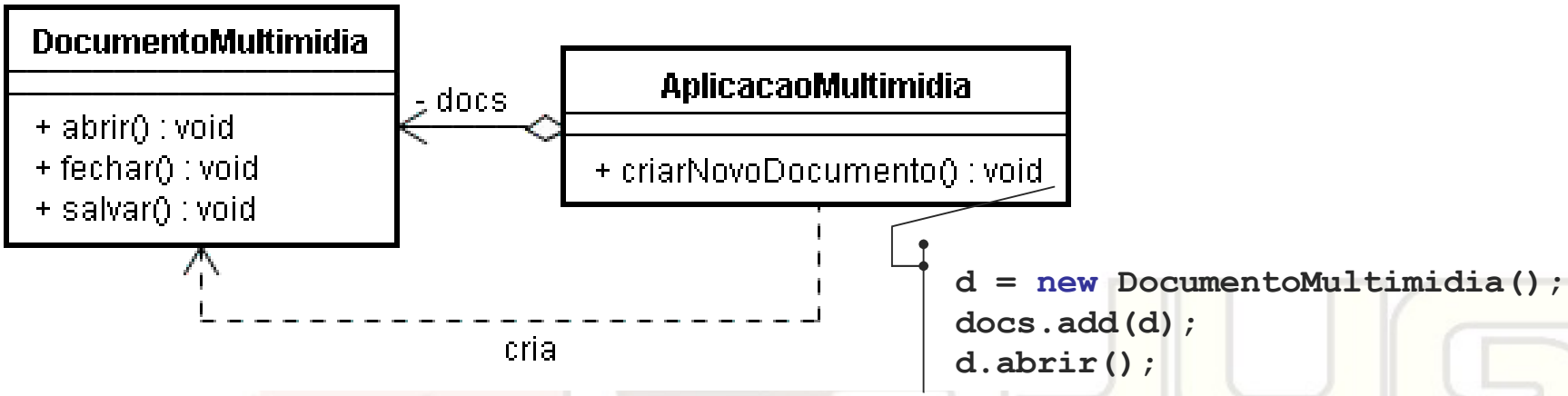
Curso - Padrões de Projeto
Módulo 2: Padrões de Criação

Factory Method
(Método Fábrica)
Criação / Classe

Descrição

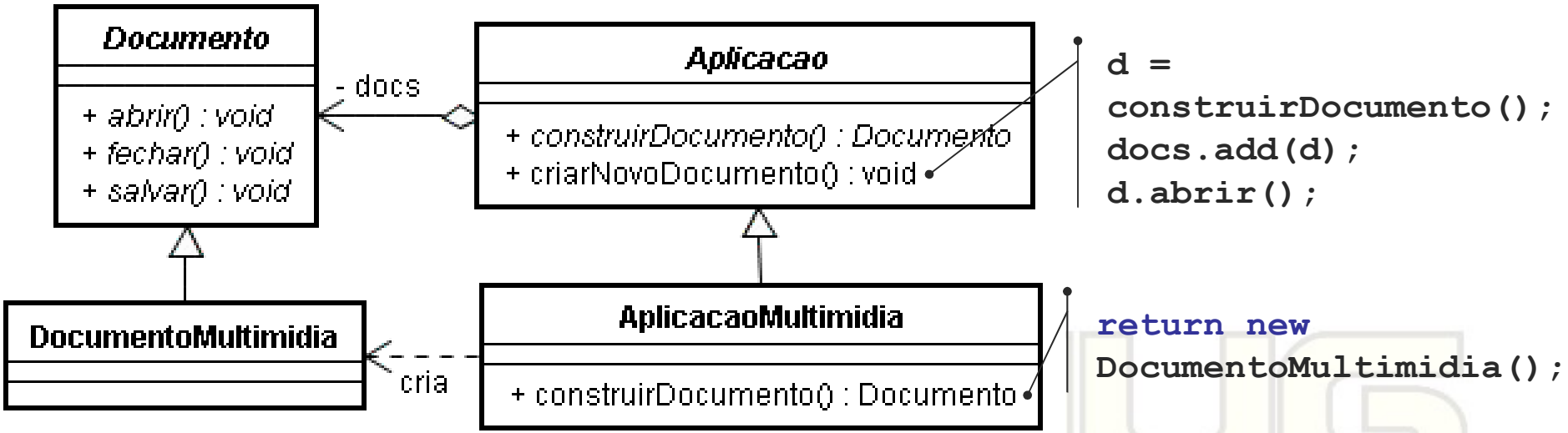
- Intenção:
  - Definir uma interface para criação de objetos mas deixar as subclasses decidirem qual classe instanciar. Em outras palavras, delega a instanciação para as subclasses.
- Também conhecido como:
  - Virtual Constructor.

O problema



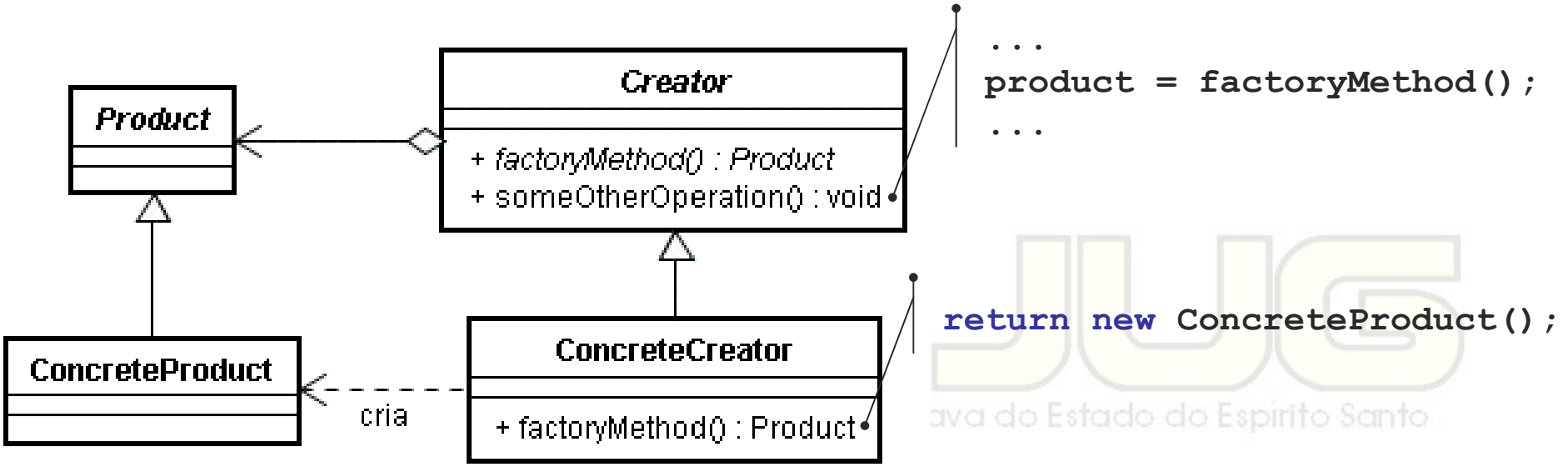
- Framework específico para uma aplicação que manipula documentos multimídia;
- É possível criar um framework mais genérico, para qualquer aplicação de manipulação de documentos?

A solução

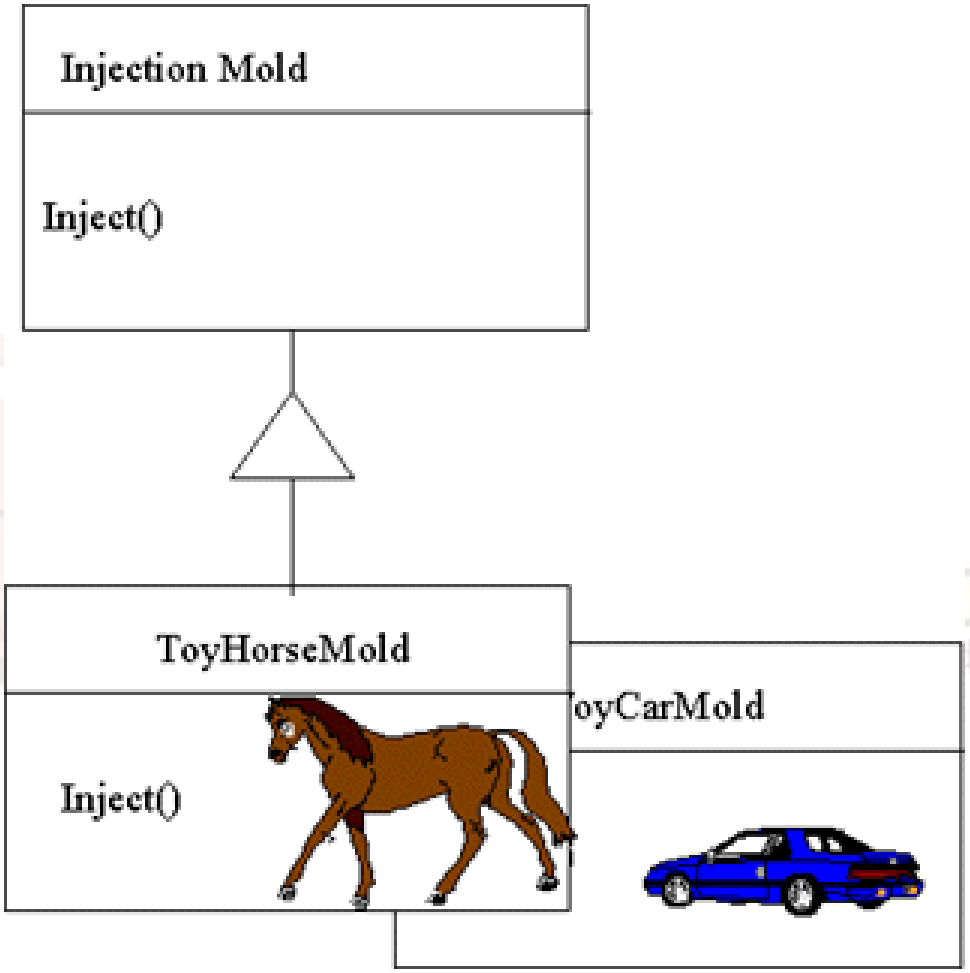


- Classes abstratas implementam as funções comuns a todo tipo de documento;
- Método fábrica é definido na superclasse e implementado na subclasse.

Estrutura



Analogia



Non-Software Examples of Software Design Patterns by Michael Duell
http://www2.ing.puc.cl/~jnavon/IIC2142/patexamples.htm

Usar este padrão quando...

- uma classe não tem como saber a classe dos objetos que precisará criar;
- uma classe quer que suas subclasses especifiquem o objeto a ser criado.

Vantagens e desvantagens

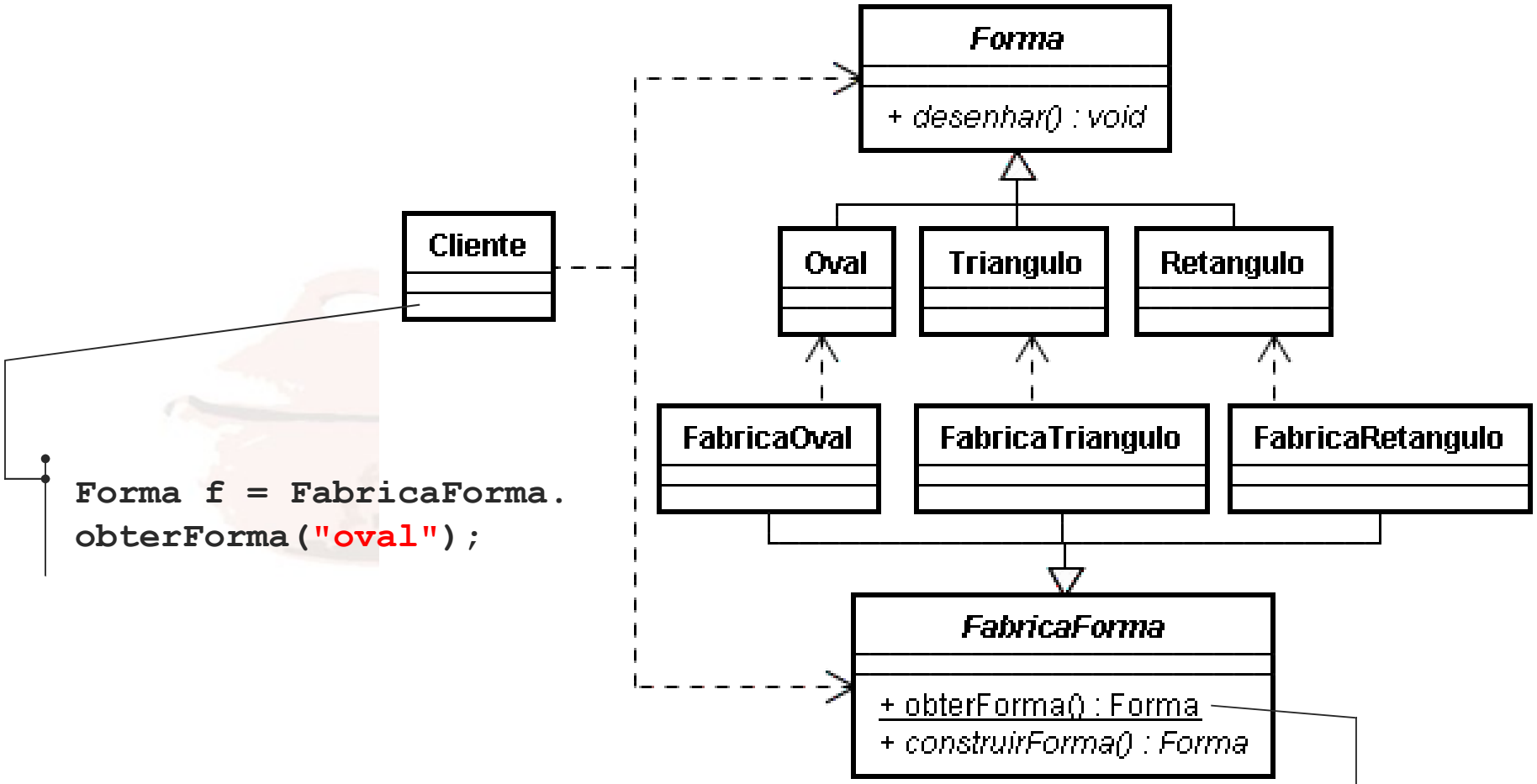
- Melhor extensibilidade:
  - Não é necessário saber a classe concreta do objeto para criá-lo.
- Obrigatoriedade da subclasse fábrica:
  - Não é possível criar somente um produto novo sem fábrica (exceto no caso do parametrizado).
- Mostraremos a seguir:
  - Extensão pela subclasse, método fábrica parametrizado e hierarquias paralelas.

Extensão pela subclasse

Retorna uma janela genérica para localizar um documento no disco e abrir.

Retorna uma janela específica para localizar documentos multimídia.

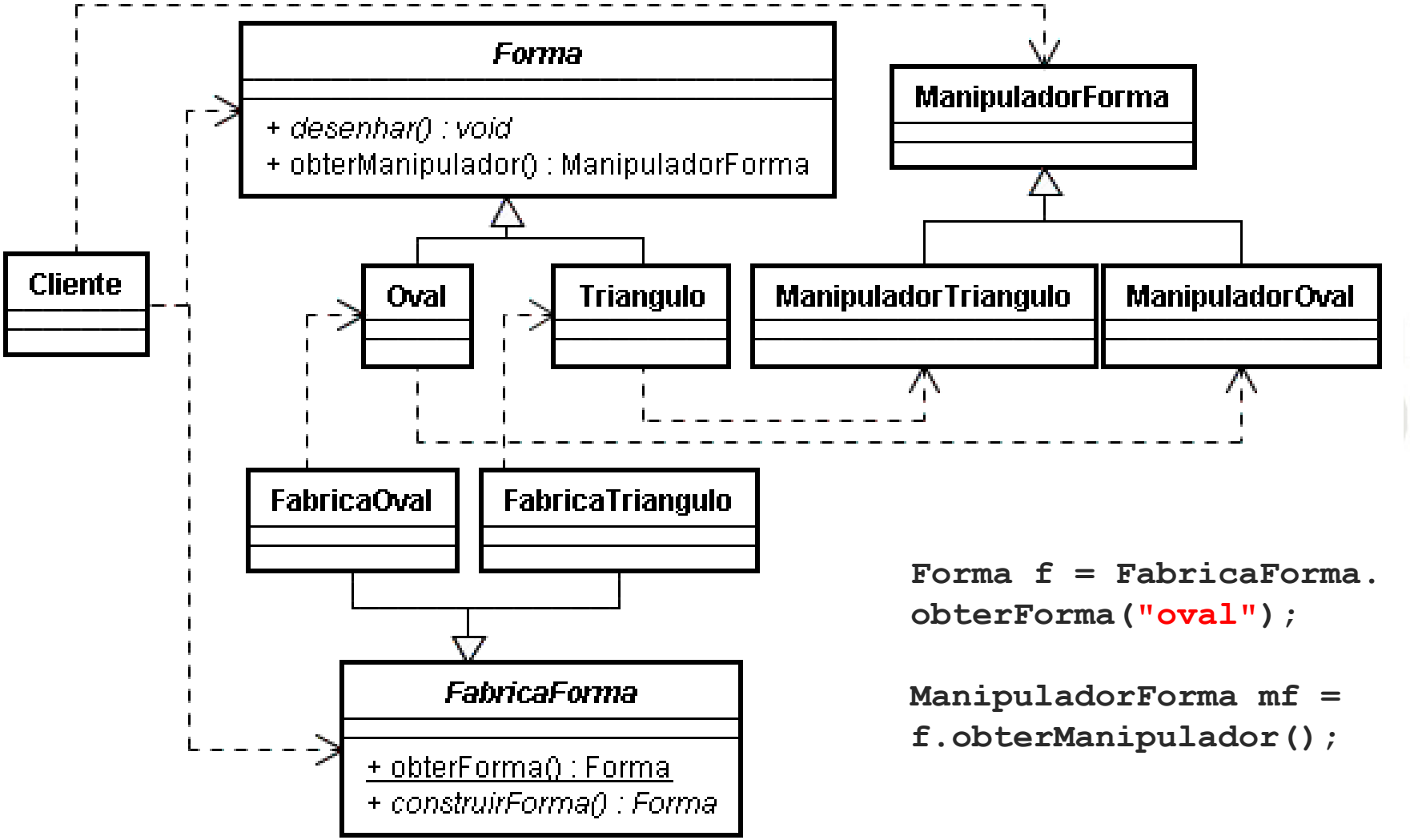
Método fábrica parametrizado




```
Forma f = FabricaForma.
obterForma("oval");
```

```
FabricaForma fab = mapaFabricas.get("oval");
return fab.construirForma();
```

Hierarquias paralelas




```
Forma f = FabricaForma.
obterForma("oval");

ManipuladorForma mf =
f.obterManipulador();
```

Curso - Padrões de Projeto
Módulo 2: Padrões de Criação

Prototype
(Protótipo)
Criação / Objeto

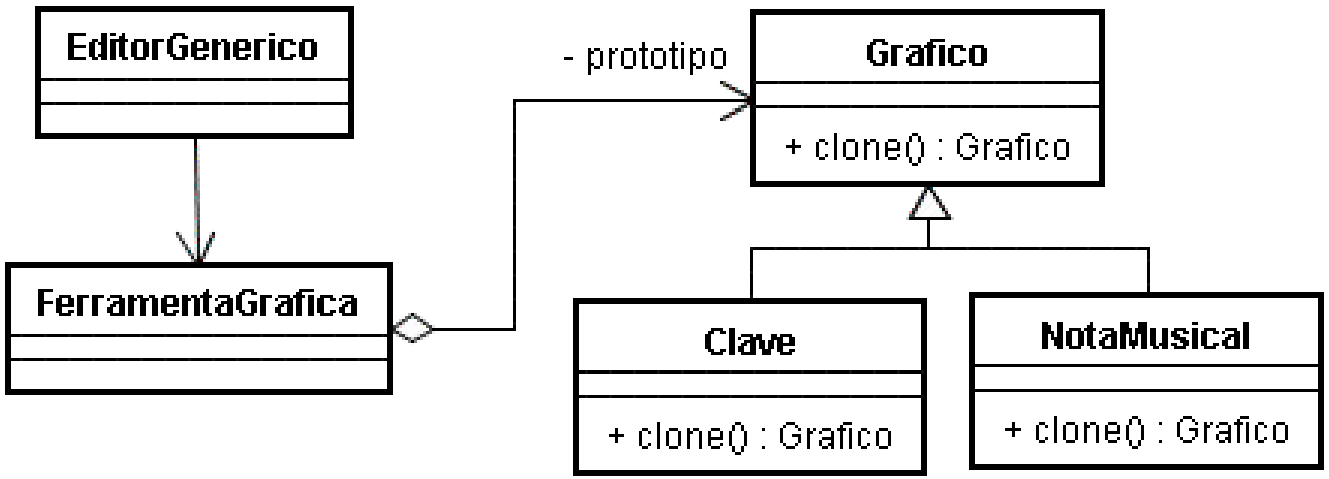
Descrição

- Intenção:
  - Especificar o tipo de objeto a ser criado utilizando uma instância como protótipo e criar novos objetos copiando este protótipo.

O problema

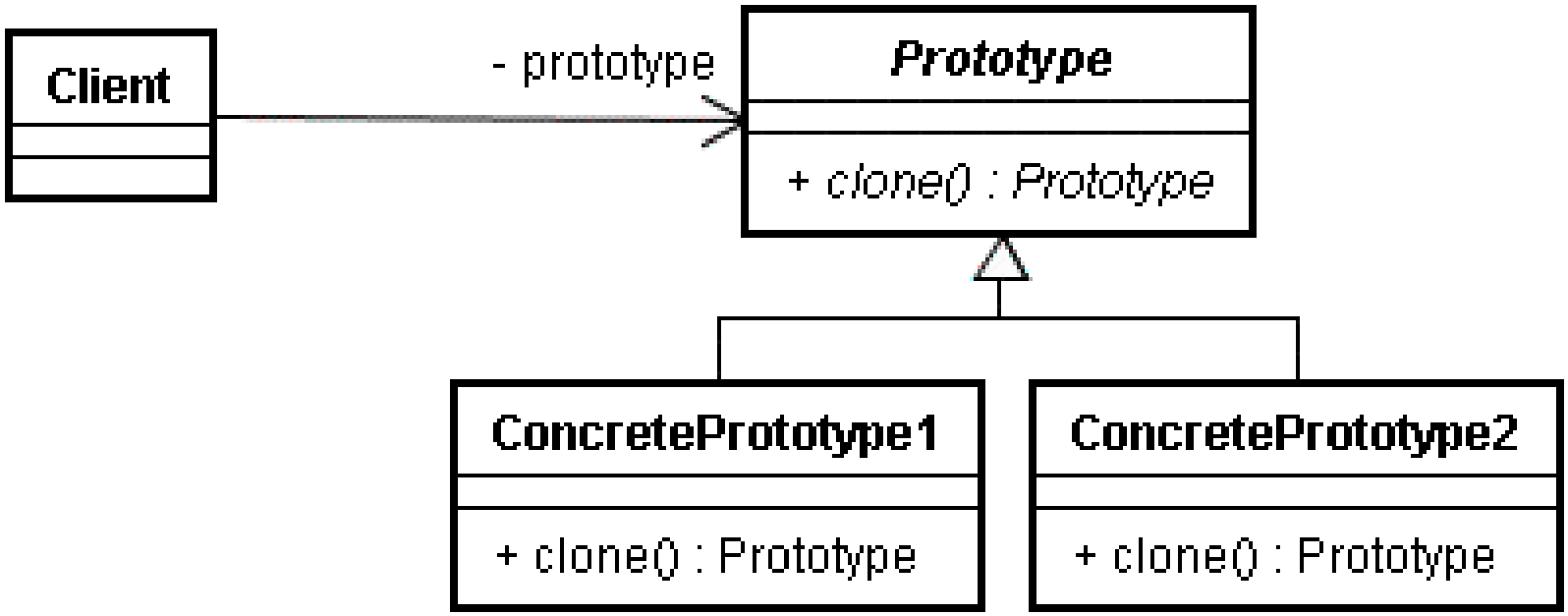
- Existe um framework para edição de documentos genéricos;
- A ferramenta gráfica para um editor musical precisa conhecer os gráficos específicos, o que anula o benefício conseguido anteriormente.

A solução

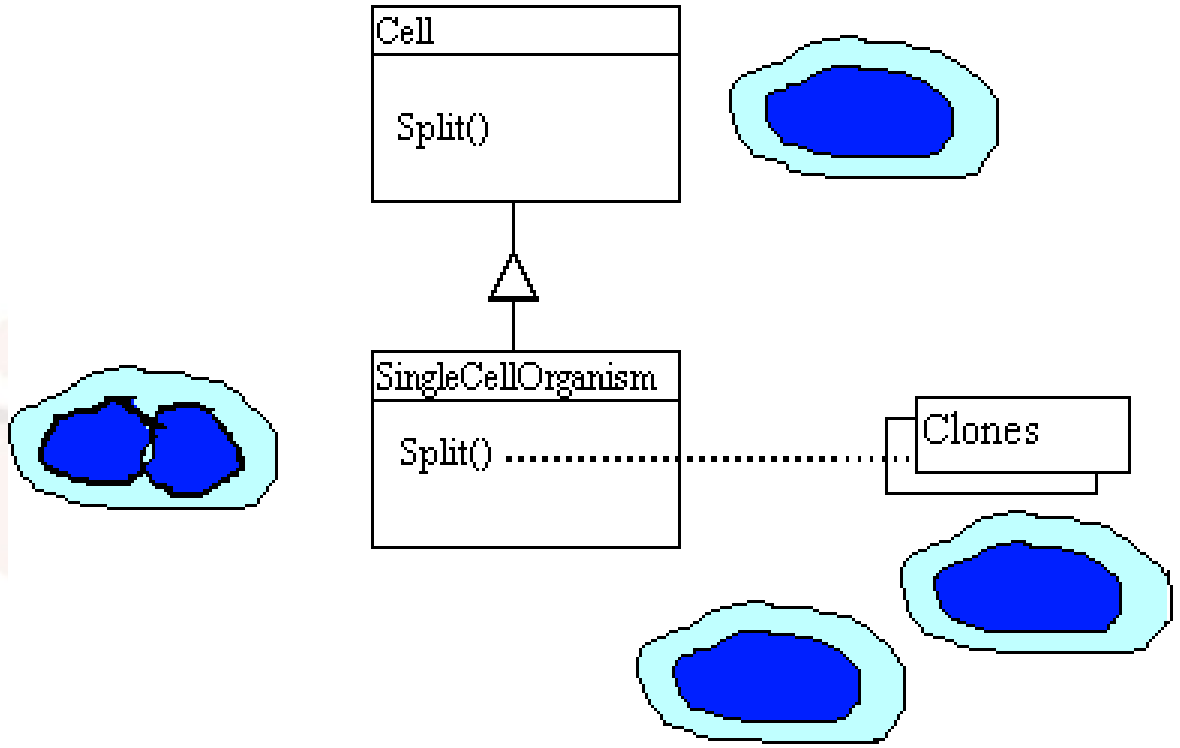


- A ferramenta gráfica é configurada com protótipos dos objetos gráficos;
- Quando precisar criar algum deles, chama o método clone(), que retorna uma cópia de si mesmo.

Estrutura


Client
- prototype
Prototype
+ clone() : Prototype
ConcretePrototype1
+ clone() : Prototype
ConcretePrototype2
+ clone() : Prototype

Analogia



Non-Software Examples of Software Design Patterns by Michael Duell
http://www2.ing.puc.cl/~jnavon/IIC2142/patexamples.htm

Prototype em Java

- Método Object.clone() é um exemplo;

```
Nota n = new Nota(Nota.SOL);
Nota copia = n.clone();
```

- Implementação:

```
class Nota implements Cloneable {
  private Tempo tempo; private int som;
  public Object clone() {
    try {
      Nota n = (Nota)super.clone();
      n.tempo = tempo.clone(); // Tempo deve ser clonável
      return n;
    }
    catch (CloneNotSupportedException e) { return null; }
} }
```

Usar este padrão quando...

- o sistema deve ser independente de como seus produtos são criados, compostos e representados e...
  - as classes que devem ser criadas são especificadas em tempo de execução;
  - ou você não quer construir uma fábrica para cada hierarquia de produtos;
  - ou as instâncias da classe clonável só tem alguns poucos estados possíveis, e é melhor clonar do que criar objetos.

Vantagens e desvantagens

- Esconde a implementação do produto;
- Permite adicionar e remover produtos em tempo de execução (configuração dinâmica da aplicação);
- Não necessita de uma fábrica para cada hierarquia de objetos;
- Implementar clone() pode ser complicado.

Curso - Padrões de Projeto
Módulo 2: Padrões de Criação

Singleton

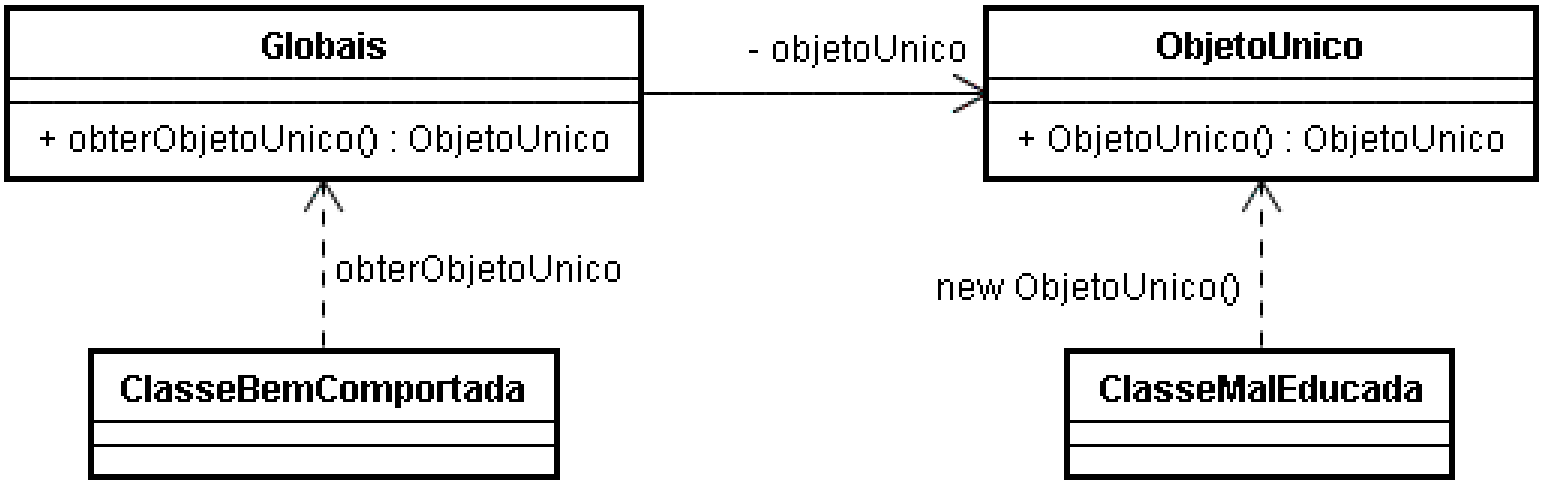
(Objeto Único)

Criação / Objeto

Descrição

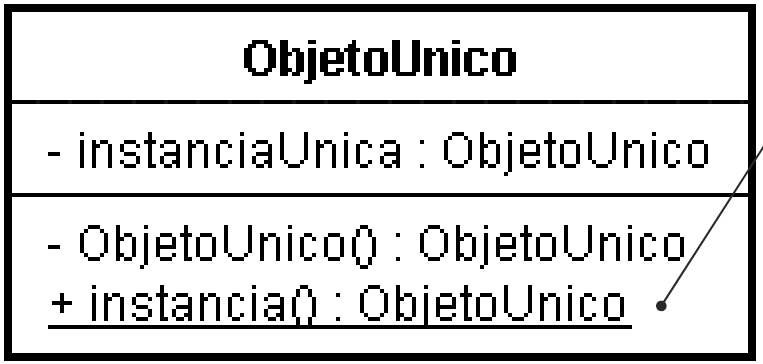
- Intenção:
  - Garantir que uma classe possui somente uma instância e prover um ponto de acesso global a ela.

O problema



- Algumas vezes você precisa que uma classe só tenha uma instância para todo o sistema;
- Prover um ponto de acesso static não é suficiente, pois classes poderão ainda construir outra instância diretamente.

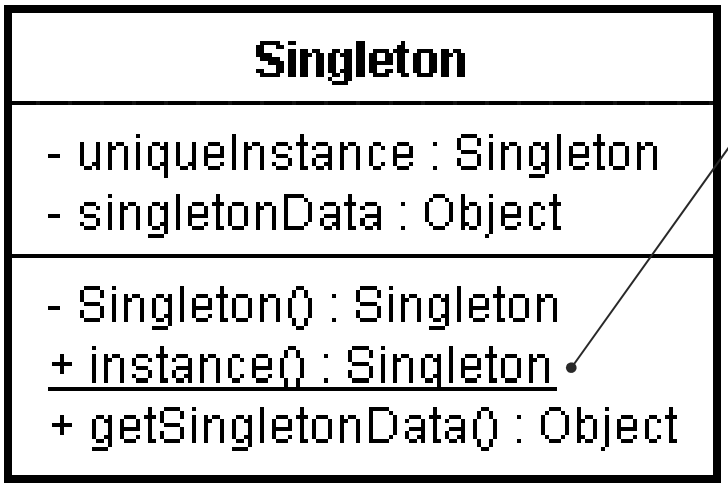
A solução




```
public static ObjetoUnico instancia() {
  if (instanciaUnica == null) {
    instanciaUnica = new ObjetoUnico();
  }
  return instanciaUnica;
}
```

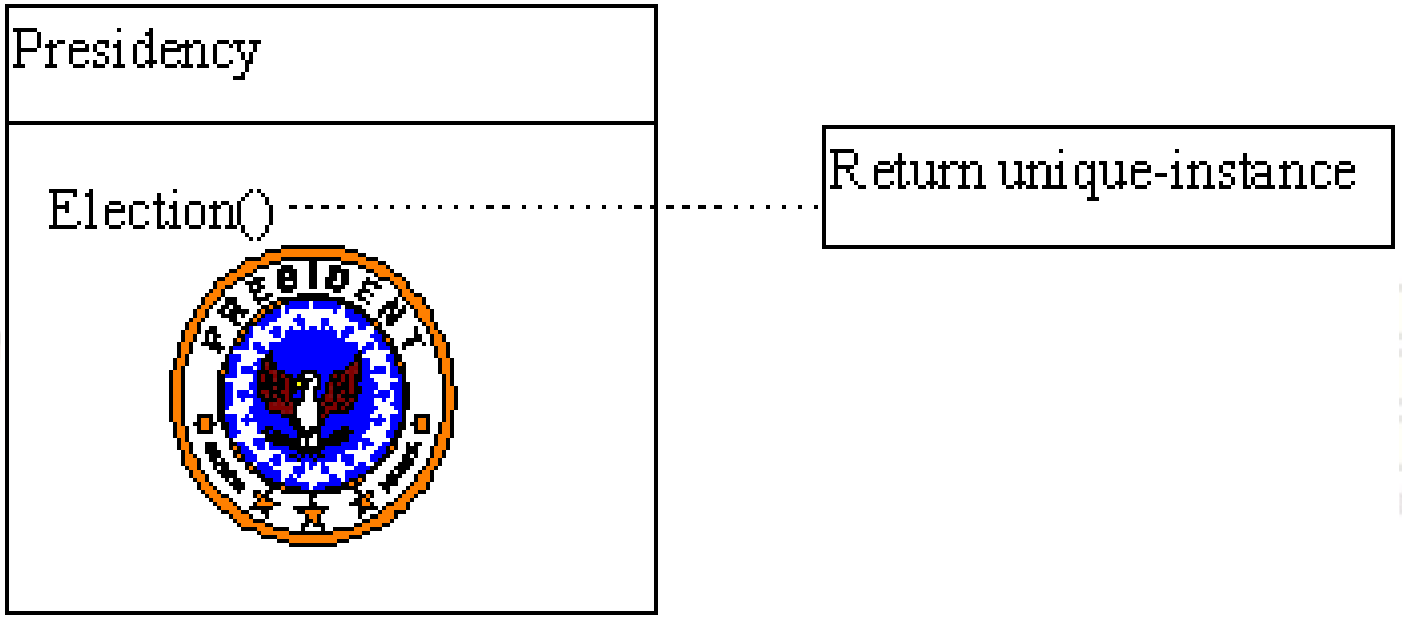
- Há um ponto de acesso global (método static);
- Construtor privado ou protected? Depende se as subclasses devem ter acesso;
- Bloco de criação poderia ser synchronized para maior seguranças em ambientes multithread;
- A instância única poderia ser pré-construída.

Estrutura




```
public static Singleton instance() {
  if (uniqueInstance == null) {
    uniqueInstance = new Singleton();
  }
  return uniqueInstance;
}
```

Analogia



Non-Software Examples of Software Design Patterns by Michael Duell
http://www2.ing.puc.cl/~jnavon/IIC2142/patexamples.htm

Usar este padrão quando...

- tiver que haver exatamente uma instância de uma classe e ela tiver que estar acessível a todos num local bem definido;
- quiser permitir ainda que esta classe tenha subclasses (construtor protected).

Vantagens e desvantagens

- Acesso controlado à instância:
  - A própria classe contrla sua instância única.
- Não há necessidade de variáveis globais:
  - Variável globais poluem o espaço de nomes.
- Permite extensão e refinamento:
  - A classe Singleton pode ter subclasses.
- Permite número variado de instâncias:
  - Você pode controlar este número.
- Mais flexível do que operações de classe:
  - Usar membro static perde flexibilidade.

Curso - Padrões de Projeto
Módulo 2: Padrões de Criação

Conclusões

Atenção na criação de objetos

- Como representar e como criar objetos é um conceito chave do seu projeto:
  - Pode facilitar ou dificultar o desenvolvimento;
  - Pode flexbilizar ou enrijecer sua estrutura de classes.
- Padrões de criação mostram diversas alternativas para criação.

Os padrões e seus usos

- Controlar o número de instâncias = Singleton;
- Processo de construção for complexo, envolvendo várias partes = Builder;
- Simplesmente flexibilizar a criação:
  - Abstract Factory: famílias de objetos;
  - Factory Method: uma fábrica por hierarquia;
  - Prototype: a fábrica é a própria classe.

Fábricas auxiliam até casos muito simples

- Métodos fábrica vs. Construtores:
  - Podem ter nomes mais significativos;
    - Ex.: BigInteger.probablePrime().
  - Não precisam criar um objeto novo toda vez;
    - Ex.: Boolean.valueOf().
  - Podem retornar instâncias de subclasses:
    - Collections.synchronizedCollection().
  - Classes sem construtores public ou protected não podem ter subclasses;
  - Não é clara a distinção entre métodos fábrica e outros métodos da classe (static).

Conclusões

- Estude os padrões:
  - Existem outras alternativas para criação;
  - Existem casos em que são necessárias;
  - Amplia sua visão como projetista.
- Atenção aos anti-padrões:
  - Singleton é considerado por alguns um anti-padrão. Argumentos:
    - Algo ter que ser global = projeto ruim;
    - Clientes ficam acoplados (dificulta os testes);
    - Viola o "princípio da responsabilidade única".

Curso - Padrões de Projeto Módulo 2: Padrões de Criação

Vítor E. Silva Souza
vitorsouza@gmail.com

http://www.javablogs.com.br/page/engenho
http://esjug.dev.java.net